timon.kelter [☺] tu-dortmund.de

Übung zur Vorlesung
Rechnerarchitektur
Sommersemester 2014

Übungsblatt 10

(10 Punkte)

Besprechung am Montag, 30. Juni 2014

10.1 Kohärenz versus Konsistenz (1 Punkt)

Beschreiben Sie jeweils mit Ihren eigenen Worten, was mit *Cachekohärenz* und *Speicherkonsistenz* gemeint ist und verdeutlichen Sie dies jeweils anhand eines einfachen Beispiels. Gibt es einen Zusammenhang zwischen Speicherkonsistenz und Cachekohärenz? Wenn ja, welchen?

10.2 Sequentielle Speicherkonsistenz (2 Punkte)

Es wird angenommen, dass drei Prozessoren P_1, P_2, P_3 eines Mehrprozessorsystems auf einen gemeinsamen Speicher zugreifen und dass das System sequentiell konsistent ist. Vor Ausführung des Programmcodes gilt $A = B = 0$. Die drei Prozessoren führen nun eine Reihe von Operationen gemäss der unten stehenden Tabelle aus. Dabei ist o_{i1} die erste Operation, die Prozessor P_i ausführt, und o_{i2} die zweite.

P_1	P_2	P_3
$o_{11} : A = 1$	$o_{21} : u = A$	$o_{31} : v = B$
	$o_{22} : B = 1$	$o_{32} : w = A$

Das Ergebnis der Ausführung kann durch das Zahlentupel (u, v, w) beschrieben werden. Dabei kann das Ergebnis variieren und zwar abhängig von der Reihenfolge, in der die Operationen der einzelnen Prozessoren global sichtbar gemacht werden. Einige der in der Tabelle unten gezeigten Tupel sind auf einem sequenziell konsistenten System nicht möglich. Füllen Sie die Tabelle folgendermassen aus (siehe auch das Beispiel in Zeile 2): Geben Sie an, ob das Tupel in der jeweiligen Zeile auf einem sequentiell konsistenten System als Ergebnis auftauchen kann und falls ja, nennen Sie eine mögliche Reihenfolge, in der die Operationen der einzelnen Prozessoren global sichtbar gemacht wurden.

u	v	w	**konsistent?**	**Reihenfolge**
0	0	0		
0	0	1	Ja	$o_{21}, o_{31}, o_{11}, o_{22}, o_{32}$
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

10.3 Speicherkonsistenzmodelle (5 Punkte)

Neben der sequentiellen Konsistenz wurden in der Vorlesung auch eingeschränktere Modelle von Speicherkonsistenz eingeführt. Betrachten Sie das folgende MIPS-Programmstück, dass von zwei Out-Of-Order Prozessoren ausgeführt wird.

	P_1		P_2
	`li $a, 0`		`li $e, 1`
o_{11}:	`sw $a, A`	o_{21}:	`sw $e, E`
o_{12}:	`lw $c, C`	o_{22}:	`aquire L`
o_{13}:	`lw $d, D`	o_{23}:	`lw $f, F`
	`add $x, $c, $d`	o_{24}:	`sw $f, R`
o_{14}:	`sw $x, X`	o_{25}:	`release L`
			`li $g, 3`
		o_{26}:	`sw $g, G`

Im Beispiel sind alle Großbuchstaben feste, jeweils verschiedene Speicheradressen. Nehmen Sie an, dass *aquire* einer *swap*-Anweisung entspricht (Lese- und Schreibzugriff auf L) und dass *release* nur einen Schreibzugriff auf L durchführt. Für diese Aufgabe sind folgende Konsistenzmodelle relevant:

- Sequentielle Konsistenz (SK)
- Total Store Order (TSO)
- Partial Store Order (PSO)
- Weak Ordering (WO)
- Release Consistency (RC)

Geben Sie für jede der folgenden Anordnungen *alle* der genannten Konsistenzmodelle an, denen Sie sie genügt. Hierbei ist es irrelevant ob eine Ausführung der Instruktionen in dieser Reihenfolge die Programmsemantik verletzt, es geht nur darum, ob die Sequenz dem jeweiligen Konsistenzmodell genügt.

Nr	**Sequenz**									
1	o_{11}	o_{12}	o_{21}	o_{13}	o_{22}	o_{23}	o_{14}	o_{24}	o_{25}	o_{26}
2	o_{11}	o_{22}	o_{23}	o_{24}	o_{25}	o_{21}	o_{12}	o_{13}	o_{14}	o_{26}
3	o_{14}	o_{12}	o_{13}	o_{22}	o_{24}	o_{23}	o_{21}	o_{25}	o_{26}	o_{11}
4	o_{12}	o_{13}	o_{21}	o_{11}	o_{22}	o_{23}	o_{14}	o_{24}	o_{25}	o_{26}
5	o_{14}	o_{12}	o_{13}	o_{21}	o_{22}	o_{23}	o_{24}	o_{25}	o_{26}	o_{11}
6	o_{14}	o_{12}	o_{13}	o_{22}	o_{23}	o_{21}	o_{24}	o_{25}	o_{26}	o_{11}

Hinweis: Es ist hierbei hilfreich zuerst einen Graphen mit der im jeweiligen Modell zu erhaltenden Reihenfolge-Relation zu zeichnen, in dem Kanten für zu erhaltende Abhängigkeiten stehen (Analog zur Vorlesung). Transitive Kanten sollten der Übersicht halber nicht eingezeichnet werden.

10.4 Locks (2 Punkte)

a) Ist die im Assemblercode des TriCore-Prozessors angegebenen Funktion eine sichere Implementierungen für das Setzen eines Locks? Begründen Sie ihre Antwort. (Eine Dokumentation zu dem TriCore-Befehlssatz finden Sie auf der Übungs-Webseite.)

`LO:var` bezeichnet die unteren 16Bit der Speicheradresse von Variable 'var'. `HI:var` steht für die oberen 16Bit der Speicheradresse von Variable 'var'.

```
lock:
      movh.a  %a12, HI:var
_loop:
      ld.w    %d8, [%a12] LO:var
      jne     %d8, 0, _loop
_success:
      mov     %d8, 1
      st.w    [%a12] LO:var, %d8
```

```
        ret
```

b) Wie kann mit Hilfe eines atomaren SWAP-Befehls, der den Inhalt eines Prozessorregisters mit dem einer Speicherstelle vertauscht, ein sicherer Lock implementiert werden?